VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2014 Tomáš Popelka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Popelka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Aktiv Communication s.r.o.
2. Struktura závěrečné zprávy:
a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c) Zvolený postup řešení zadaných úkolů
d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Marek Menšík, Ph.D.**

Konzultant bakalářské práce: Ing. Ondřej Rychtář

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 23. 4. 2014 .....

Rád bych tímto poděkovat firmě Aktiv Communication s.r.o. a hlavně zejména panu Ing. Ondřeji Rychtářovi za umožnění absolvování mé bakalářské praxe v této firmě.
Dále bych chtěl rád poděkoval svému vedoucímu Bakalářské práce panu Mgr. Marku Menšíkovi, Ph.D. za jeho poskytnuté konzultace k obsahu této práce.

Abstrakt

V této bakalářské práci je popsána má činnost ve firmě Aktiv Communication s.r.o. Nejdříve se zaměřím na popis samotné firmy a poté přejdu k jednotlivým úkolům a jejich řešením, které jsem v této praxi vykonal. Na závěr provedu shrnutí mé praxe z pohledu získaných zkušeností a znalostí.

Klíčová slova: Bakalářská praxe, PHP, MySQL, Dibi, Nette, SOAP

Abstract

In this bachelor is described my activity in the company Aktiv Communication s.r.o. First I focus on the description of the company itself and then go to each of the tasks and their solutions which I accomplished in this practice. At the end I make a summary of my experience from the perspective gained experience and knowledge.

Keywords: Bachelor practice, PHP, MySQL, Dibi, Nette, SOAP

Seznam použitých zkratek a symbolů

PHP	–	Hypertext Preprocessor
CSS	–	Cascading Style Sheets
JS	–	JavaScript
AJAX	–	Asynchronous JavaScript and XML
SOAP	–	Simple Object Access Protocol
XML	–	Extensible Markup Language
UI	–	User Interface
OOP	–	Object-Oriented Programming
API	–	Application Programming Interface
MVP	–	Model View Presenter
VPS	–	Virtual Private Server
WSDL	–	Web Services Descritpion Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Jako způsob uskutečnění mé bakalářské práce jsem si vybral absolvování odborné praxe ve firmě místo vypracování klasické bakalářské práce. Rozhodl jsem se pro to z několika důvodů. Chtěl jsem si vyzkoušet práci ve firmě a porovnat mé představy s tím, jak to skutečně probíhá. Také si myslím, že se tímto způsobem dá získat mnoho nových znalostí, které bych jinak nezískal.

Pro firmu Aktiv Communication s.r.o. jsem se rozhodl, protože v době výběru praxe jsem byl již u ní zaměstnán. V této firmě jsem začal pracovat po té, co mě oslovil její majitel, jestli bych neměl zájem jim vytvořit nový eshop kompletně na míru. Ihned po první schůzce se ukázalo, že se zdaleka nejedná jenom o jeden eshop, ale o kompletní systém, který by umožňoval spravovat několik na sobě zcela nezávislých eshopů. Tyto eshopy nemusí být jen firemní, ale i od případných klientů. Sortimemnt eshopů jsou převážně hry, ale i nějaký software. V obou případech se jedná o tzv. digitální distribuci, což je systém, kdy se prodá jenom registrační kód k danému produktu a samotný produkt se následně stáhne z internetu. Neprodávají se žádné fyzické média.

2 Zaměření firmy a mého pracovního nasazení

2.1 Popis firmy

Původně jsem začal pracovat ve firmě Techno OVA s.r.o. Tato firma se zaměřuje na výrobu a úpravu lihu. K dnešnímu dni byla zrealizována výstavba více než dvaceti rafinérií lihu v České republice a na Slovensku.

V roce 2012 firma Techno OVA s.r.o. začala provozovat svůj první eshop s licenčními klíči. Jednalo se o zcela jiný druh podnikání. Firma Aktiv Communication s.r.o. vznikla v létě roku 2013 odloučením části firmy Techno OVA s.r.o., která se zabývala provozem eshopu Key4You.cz. Nyní firma Aktiv Communication s.r.o. nabízí pronájem eshopů pro prodej licenčních klíčů ke hrám a softwaru. Klient si může případně objednat i vytvoření eshopu na míru. Jedná o tzv. digitální distribuci, kde koncový zákazník získá pouze aktivační/registrační/licenční číslo/kód a samotnou aplikaci či hru si musí již sehnat sám. Většinou lze využít platforem jako jsou Steam, Origin atp. Dále firma zprostředkovává zásobování těchto pronajatých eshopů licenčními klíči. Také zajišťuje technickou podporu pro jednotlivé eshopy.

2.2 Pracovní zařazení

V době mého nástupu měla firma již funkční eshop key4you.cz, ale nebyla spokojena s možností jeho administrace a také chtěla rozšířit a hlavně usnadnit napojování dalších eshopů na společnou administraci.

Mým úkolem tedy bylo ještě spolu s jedním kolegou, který má na starost databázovou část celého systému, navrhnout a následně vytvořit celý nový systém. Kromě nás dvou na tomto systému nikdo jiný nepracoval, což byla pro mě docela výzva. Již jsem měl zkušenosti s několika menšími projekty, ale nikdy jsem do této doby nepracoval na něčem takovýchto rozměrů.

3 Starý systém

Celý systém byl postavený na technologiích, které jsou zdarma. Jako operační systém byl využit Linux Cent OS 6.0. Za webový server byl zvolen Apache 2.0[4], se kterým spolupracovalo PHP 5.2 a MySQL 5.1[3]. Pro správu databáze bylo využíváno rozhraní phpMyAdmin. Na tomto serveru dále běžely služby jako mailserver a DNS server[5].

Ve starém systému se původně nepočítalo s napojováním více eshopů na společnou administraci či sklad. Jeho rozšiřování bylo prakticky nemožné, protože způsob jeho implementace byl naprosto chaotický. Jeho autor nevyužíval žádných dnes již obvyklých standardů jako je OOP, ani rozumného oddělování bussiness logiky od prezenční vrstvy.

Jak již bylo uvedeno výše, starý systém původně nepočítal s rozšiřováním a napojováním dalších eshopů. Také tomu odpovídala struktura databáze. Nebylo zde pamatováno na různé jazykové verze textů, ani na odlišné nastavení parametrů či cen jednotlivých eshopů. Také se strukturou jednotlivých tabulek nebyla firma spokojena, proto chtěla celou databázi navrhnout znovu.

Pro kompletně vlastní systém jsme se rozhodli proto, že opensourcové systémy na trhu, které byly zdarma, nesplňují všechny naše požadavky. Hlavně ve všech těchto systémech chyběla možnost administrace více na sobě nezávislých eshopů, které by ale využívaly společného skladu. Dalším důvodem pro zcela vlastní systém bylo to, že se počítalo s mnoha změnami, které se budou v budoucnu provádět a vždy je lepší upravovat vlastní systém, než se snažit něco napojovat tam, kde se s tím nikdy nepočítalo. Posledním faktorem pro vlastní řešení byla samozřejmě cena. Vytvoření vlastního systému je mnohem levnější, než využití nějaké placené varianty. Po zvážení těchto faktorů nebylo nad čím uvažovat a nový vlastní systém byla jasná volba.

4 Nový systém

Protože jsem celý nový systém měl začít tvořit od základu, tak jsem si mohl také vybrat technologie, které budu využívat. Protože byl již nakonfigurovaný server, tak jsem se rozhodl v tomhle směru nic neměnit. Takže technologie běžící na serveru zůstaly stejné jako v případě starého systému - PHP, MySQL.

Pro usnadnění a urychlení práce jsem si vybral PHP framework Nette 2.0. Jedná se framework, který vznikl u nás v České republice a který má za sebou velice silnou komunitu hlavně českých uživatelů. Také má velice dobrou dokumentaci [1]. Dnes se používá ve většině českých firem, které se zabývají vývojem webů pomocí PHP. Samozřejmostí je dodržení základního rozdělení aplikace v Nette stylu, a to MVP. Jedná se o stejné dělení jako v případě návrhového vzoru MVC, jen je dodržena Nette terminologie - Presenter.

Pro zjednodušení tvorby SQL dotazů a jejich zabezpečení jsem se rozhodl použít Dibi, což je abstraktní vrstva pro PHP 5.x, která se právě řešením těchto problémů zabývá.

Pro vytvoření více user friendly UI bylo využito CSS 3 a jQuery[6] spolu s jQuery UI.

4.1 Architektura systému

Další požadavek byl, aby mohly jednotlivé části celého systému běžet na různých serverech. Částmi se zde myslí frontend (samotný eshop), administrace a databáze. A také zabezpečení přístupu k databázi pro koncové frontendy v případě, že by chtěl klient ehsop provozovat na vlastním hostingu. Protože v tomto případě by měl přístup ke zdrojovým kódům a mohl by tak získat přístup k databázi, rozhodli jsme se pro oddělení těchto vrstev pomocí API. Pro tento účel jsme zvolili technologii SOAP[8], která nám umožňuje vyměňovat data ve formátu XML[7] přes HTTP protokol. Díky tomu lze dynamicky pracovat s daty na frontendu, aniž by ve zdrojových kódech byl někde uveden přístup k databázi. Každý eshop získá své APIName a APIPassword. Díky tomu lze zajistit, že frontendy mohou pracovat pouze se svými daty. Aktuálně běží administrace a databáze na společném firemním VPS. Samotný frontend eshopu pak na svém vlastním serveru/hostingu. (Rozložení celého systému lze vidět na obr. 1)

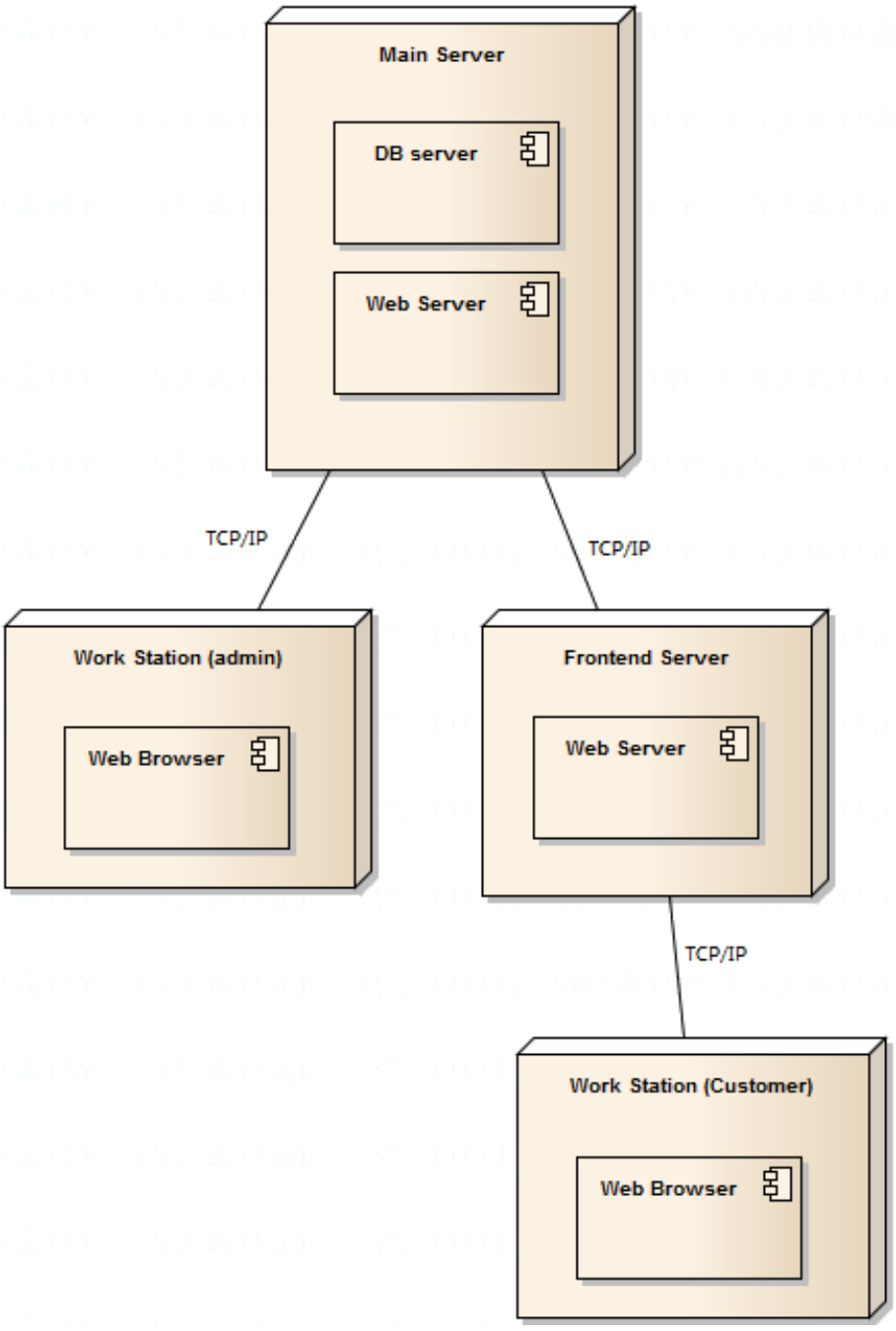
4.2 Nová administrace

Administrace byla rozšířena o spousty nových funkcí jako snadnější možnosti filtrování jednotlivých seznamů, ať už jde o produkty, zákazníky, licenční klíče či čehokoliv dalšího.

Také bylo kompletně změněno její UI a přizpůsobeno novým funkcím, hlavně pro zadávání různých jazykových mutací pro jednotlivé texty a rozlišení jednotlivých eshopů.

Další významnou změnou administrace bylo přidání možnosti vytvářet nové uživatele, kterým lze nastavovat rozdílné oprávnění v jednotlivých eshopech.

Pro umožnění využívání všech těchto nových a upravených funkcí byla vytvořena zcela nová databáze.



Obrázek 1: Architektura nového systému

4.3 Nová databáze

Hned na začátku bych rád podotkl, že strukturu nové databáze jsem nenavrhoval já, ale kolega. Jak již bylo uvedeno výše, starý systém původně nepočítal s rozšiřováním a napojováním dalších eshopů. Také tomu odpovídala struktura databáze. Nebylo zde pamatováno na různé jazykové verze textů, ani na odlišné nastavení parametrů či cen jednotlivých eshopů. Také se strukturou jednotlivých tabulek jsme nebyli spokojeni, tak jsme se rozhodli celou databázi navrhnout znovu.

Pro každou entitu, která obsahuje jazykově závislé texty, byla vytvořena speciální tabulka ve tvaru *entitaT*, která je propojena s hlavní tabulkou pro entitu pomocí jejího primárního klíče. Dále je v této tabulce součástí primárního klíče navíc ještě sloupec, který označuje jazykovou verzi daného záznamu. Tímto způsobem jsme schopni velice jednoduše vytvářet nové jazykové verze všech textů.

Další změnou je u většiny entit rozlišení, kterému eshopu náleží a to pomocí rozšíření jejich primárního klíče. Tento sloupec následně uchovává název eshopu, ke kterému daný řádek tabulky patří.

V neposlední řadě jsme se rozhodli pro zaznamenávání všech změn, které kdokoliv provedl v administraci. Za tímto účelem vznikla tabulka *changeLog*, kde se zapisují staré a nové hodnoty, tabulka a primární klíč od záznamu, kterého se změna týká, také datum a čas provedení změny a jméno uživatele, který tuto změnu provedl. Primární klíč změněného záznamu se ukládá ve formátu JSON. Tato tabulka slouží hlavně k řešení případných problémů, ale také k potvrzení kdo, kdy, co nastavil.

Ukázku struktury databáze lze najít v příloze A.

5 Úkoly

5.1 Seznámení

Nejdříve ze všeho mi byl představen stávající systém a byl jsem seznámen s novými požadovanými funkcemi.

5.1.1 Uživatelé a zákazníci

S celým systémem mohou pracovat čtyři typy uživatelů.

Prvním z nich je uživatel (administrátor), který má za úkol správu jednotlivých eshopů pomocí administrace. Každý uživatel může spravovat jen ty eshopy a nastavovat ty možnosti, ke kterým má povolena přístupová práva. Existuje tu jedna výjimka a to je účet pro majitele celého systému, který může spravovat všude všechno a nelze ho nijak omezit pomocí přístupových oprávnění.

Druhým typem uživatele je běžný zákazník. Toto jsou všichni nově zaregistrovaní zákazníci.

Dalším typem jsou členové +klubu. Tímto členem se může stát libovolný zákazník po koupi členství na straně frontendu. Pro tyto zákazníky se následně používají zvýhodněné ceny při nákupech. Jak zvýhodněné je záležitostí konkrétního eshopu. Stejně tak je otázkou eshopu, zda tuto skupinu zákazníků chce rozlišovat či nikoliv.

Poslední skupinou jsou velkoobchodníci. Pokud se někdo chce stát velkoobchodníkem, tak už musí sám kontaktovat majitele eshopu a domluvit se s ním na určitých podmínkách. Těmto zákazníkům jsou následně nabízeny ještě zvýhodněnější ceny, než obyčejným zákazníkům či členům klubu. Opět je záležitostí eshopu, zda tento typ zákazníků podporují.

Je tu ještě jeden typ uživatele a to tzv. API. Nejedná se o uživatele v pravém slova smyslu. Nestojí za ním žádný skutečný člověk, ale je používán při automatickém vyřizování plateb pomocí některých platebních metod, které tento způsob podporují.

5.1.2 Cenové hladiny

Pro každý produkt existují 4 cenové hladiny. První je nákupní cena, která slouží pouze pro pozdější vytváření statistik a kontroly, že se produkty prodávají stále se ziskem. Tuto informaci uvidí pouze uživatelé administrace, kteří na to mají dostatečné oprávnění.

Další cenová hladina je určena pro zadávání cen pro běžné zákazníky. Toto odpovídá cenám na většině dnešních eshopech.

Třetí hladinou je členská cena. Tato cena se počítá buď pomocí nějakého předem nastaveného koeficientu nebo je také možné nastavit ji u každého produktu ručně dle libosti.

Posledním typem cen jsou velkoobchodní ceny. Tyto ceny jsou určeny velkoobchodníkům. Stejně jako v případě členské ceny je velkoobchodní cena počítána automaticky pomocí předem nastaveného koeficientu, případně lze tuto cenu nastavit ručně.

5.1.3 Benefit (věrnostní) body

Každému zákazníkovi se za každou objednávku přičtou benefit body. Množství benefit bodů záleží na nastaveném koeficientu. Dalším faktorem, který může ovlivnit množství bodů získaných za objednávku, je typ zákazníka. Pro členy lze nastavit jiný koeficient než pro běžné zákazníky. Benefit body lze uplatnit k získání slevy při objednávce nebo k zaplacení celé objednávky. Velkoobchodníci ale žádné body nedostávají. Zda eshop bude využívat benefit bodů opět záleží na jeho nastavení.

5.1.4 Slevové akce

Bylo požadováno, aby systém uměl zpracovávat dva druhy slevových akcí - standardní sleva a tzv. rychlovka. Obě akce jsou určeny pro všechny zákazníky. Rozdíl je v tom, že v případě rychlovky lze v jedné objednávce koupit pouze jeden kus zboží v rychlovce. Další odlišnosti jsou až na straně frontendu. U jednotlivých typů slevových akcí je štítek, na kterém je uvedeno, o jaký typ slevy se jedná. V případě rychlovky běží časový odpočet do konce akce.

5.1.5 Platební metody

Další části systému je podpora různých způsobů plateb. Jsou využívány platební systémy od společností PayU, GoPay, PayPal, TrustPay, SMS, SMS SK, PaySafeCard, benefit body a také objednávky z Aukro. Většina těchto plateb je vyřizována automaticky (pokud to je možné). V administraci následně lze zjistit, jakým způsobem byla objednávka zaplacena.

5.2 Návrh databáze a modelu

Samotnou databázovou strukturu jsem netvořil já, nýbrž kolega. Já jsem pouze následně podle ní vytvořil doménový model administrační části. V této části jsem s kolegou několikrát konzultoval strukturu databáze a také jsme několikrát během její tvorby spoustu věcí předělávali dle aktuálních jednotlivých požadavků majitele systému. Ukázku struktury databáze lze najít v příloze.

5.2.1 Entity

Jednou z prvních informací, kterou jsem se dozvěděl, bylo, že mám počítat s tím, že jednotlivé části systému mohou běžet na různých serverech. Částmi systému zde bylo myšleno frontend, administrace a databáze. V této části je podstatná část administrace a databáze. Protože posílání dat odněkud někam je poměrně časově náročné, zvlášť v případě, že se nejedná o stejnou místní síť, rozhodl jsem se mít všechny její vlastnosti uloženy v každé entitě dvakrát. Jednou to jsou původní hodnoty získané z databáze a podruhé aktuální hodnoty nastaveny uživatelem. Tak lze posílat v sql dotazech jen změněná data, čímž se sníží časová náročnost přenosu. Díky tomuto řešení ovšem vzrostla paměťová náročnost jednotlivých entit prakticky na dvojnásobek. Jelikož je přikoupení paměti v současné době poměrně levnou záležitostí, rozhodli jsme se jít touto cestou.

Díky výše zmíněnému způsobu ukládání dat má každá entita dvě funkce - setOldProperties() a getChanges(). Funkce setOldProperties je volána ihned po sestavení entity a má za úkol zkopírovat jednotlivé vlastnosti do jejich podtržítkem prefixovaných verzí. K těmto prefixovaným proměnným nelze přistupovat z vnějšku - jsou private. Funkce getChanges je volána před uložením dat do databáze a vrátí jen změněné vlastnosti entity ve formě asociativního pole. Logika této funkce je velice jednoduchá. Projde všechny podtržítkem prefixované proměnné daného objektu a porovná jejich hodnoty s jejich neprefixovanými verzemi. V případě nerovnosti přidá do asociativního pole novou hodnotu s klíčem, který odpovídá názvu proměnné, který odpovídá názvu sloupce v databázi.

```
public function getChanges() {
    $changes = array();

    foreach (get_object_vars($this) as $k => $v) {
        if ($k[0] === '_') {
            $tmp = substr($k, 1);

            if ($this->$k !== $this->$tmp)
                $changes[$tmp] = $this->$tmp;
        }
    }

    return $changes;
}
```

Výpis 1: Funkce getChanges()

Toto pole se, v případě že není prázdné, potom díky Dibi velice jednoduše použije při sestavování update SQL dotazu.

```
$this->connection->update('product', $data)
                        ->where('productId = %i', $product->getProductId())->execute();
```

Výpis 2: Sestavení update SQL dotazu

Třídní diagramy entit lze najít v příloze B.

5.2.2 Repository a Mapper

Protože hned od začátku se vědělo, že systém se bude nadále rozvíjet a tudíž měnit databázová struktura, tak bylo nutné oddělit od sebe logický přístup k datům od skutečné databázové struktury. K tomu byly použity třídy Repository a Mapper a jejich verze s názvem tabulky v prefixu.

Repository slouží pro přístup k datům z aplikace. Sám o sobě neví nic o struktuře uložených dat v databázi. Pouze diktuje, jaké podmínky mají být splněny, aby se získala potřebná data. Těmto datům poté nastaví pomocí funkce setOldProperties "staré"hodnoty a případně pomocí dalších repositářů získá další data, která jsou logicky vázaná a také potřebná k plnohodnotnému používání dané entity. V některých případech není vždy nutné získávat všechny informace, ale vystačíme si jen s částí. K tomu souží funkce

setInstances, kde se posílají bool hnodnoty, pomocí kterých se rozhodne o tom, jaká data se budou potřebovat. Repository také řeší cacheování dat v rámci jednoho requestu. Pokud například chceme získat platformy ke 100 produktům a k některému produktu se má přiřadit platforma, která již dříve byla načtena, tak nedojde ke zbytečně opakovanému požadavku na databázi.

Mappery slouží pro samotné poskládání SQL dotazu na základě podmínek získaných od Repository. Tento dotaz se následně posílá na databázi. Také se zde provádí logování všech provedených změn v případě jakéhokoliv update dotazu do changeLog tabulky. Při získávání dat se samotná data mapují na konkrétní objekty, které potom jsou vráceny zpátky Repository.

5.3 Autentifikace a authorizace uživatelů

Systém má umožňovat více lidem spravovat více eshopů, ať už každému svůj nebo i více lidem jeden eshop. Je důležité si tady uvědomit, že ne všechny informace jsou určeny pouze pro konkrétní eshop. Mnoho jich ovlivňuje všechny eshopy - například lokalizace produktů nebo screeny a videa k jednotlivým produktům. Proto ne všechna data mohou být editována ze všech eshopů. Další podstatným požadavkem bylo, že ne všichni uživatelé jednoho eshopu si mají být rovni - každý by měl mít jiná oprávnění.

Nastavení oprávnění jednotlivým uživatelům se provádí tak, že každému uživateli se v každém eshopu, který má mít možnost spravovat, přiřadí nějaká role.

Každá role obsahuje seznam míst, do kterých má uživatel přístup - většinou se jedná o jednotlivé položky menu.

Ke každému oprávnění lze navíc přidat autorizační objekty, které umožňují nastavování jemnějšího oprávnění.

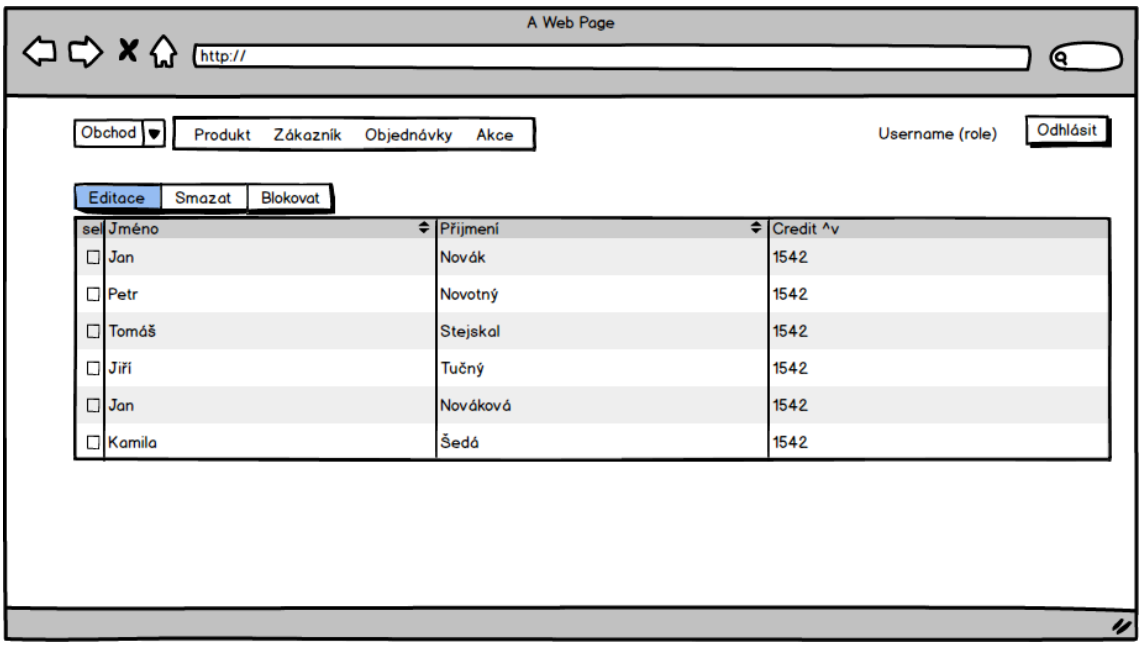
Například chceme uživateli nastavit přístup k seznamu zákazníků, ale nechceme mu umožnit jejich editaci. Nastavíme oprávnění pro přístup do Zákazníci - Seznam, ale nepřidělíme autorizační objekt pro editaci.

5.4 Návrh a tvorba GUI administrace

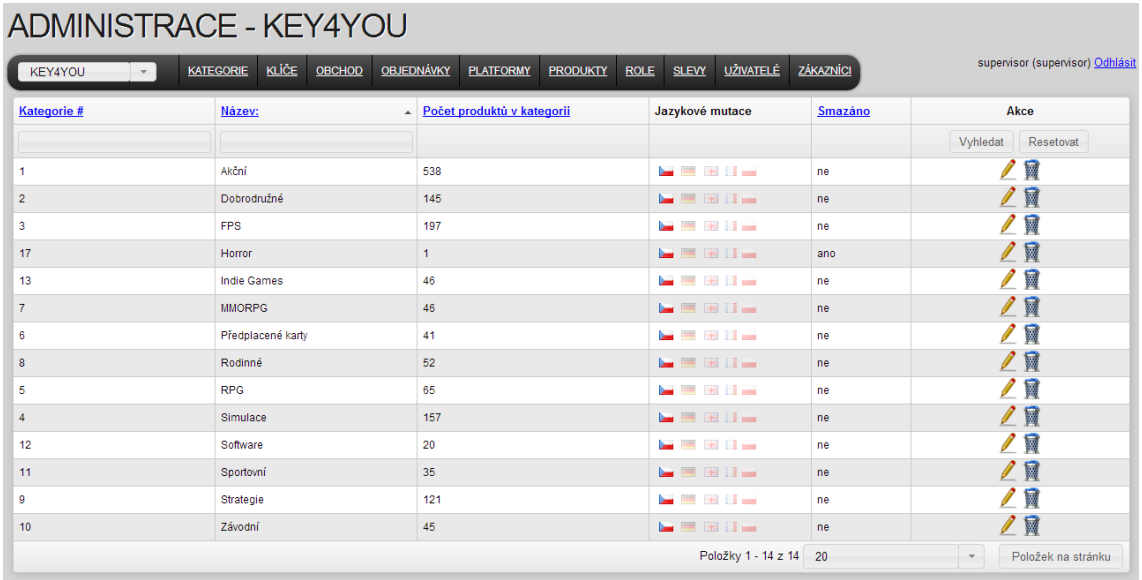
Pro usnadnění tvorby administrace bylo použito java scriptových frameworků jQuery a jQueryUI.

Dlouho jsme hledali vhodný datagrid, který by splňoval všechny naše podmínky, až jsme se rozhodli pro Grido. Jedná se o addon do Nette frameworku od člena komunity. Jedná se o komponentu, která umožňuje přehledný výpis dat, jeho formátování, řazení, stránkování, filtrování, hromadné akce a také export dat (ten však nebyl využit). Jeho implementace je velice jednoduchá, hlavně díky tomu, že je určen přímo pro Nette a jako jedním ze zdrojů dat podporuje námi používané Dibi. Pouze se vytvoří objekt, kterému se předá DibiFluent objekt, který je sestaven z obecných požadavků pro získání dat. Jednotlivé akce jsou jednoduše navěšeny pomocí callbacků, takže se získávají z databáze skutečně jen ta data, které se zrovna mají zobrazit. Návrh GUI nové administrace je k vidění na obr. 2 a finální podoba na obr. 3.

Ovšem narazili jsme na jednu ”překážku”a to získávání celkového počtu dat (nutné k sestavení stránkování). Tento grid si tuto informaci obstarává sám a to tak, že skutečný SQL dotaz pro získání dat obalí novým dotazem, který spočítá počet vrácených záznamů z prvního dotazu. Bohužel MySQL nepodporuje využívat indexů nad tabulkami ve vnořených dotazech, takže v některých případech toto bylo velice pomalé. Kvůli tomu se musela část Grida přepsat. A to konkrétně tak, že při tvorbě instance Grida lze nastavit co použít jako parametr pro funkci count v sql dotazu. Tato část následně nahradí původní select.



Obrázek 2: Návrh GUI adminsitrace



Obrázek 3: Finální podoba GUI administrace

6 API

Pro tvorbu komunikace echopu s databází jsme se rozhodli pro technologii SOAP. Jedná se o protokol pro posílání XML zpráv pomocí HTTP či SMTP protokolu. V našem případě se ovšem využívá pouze HTTP protokol. Pro jeho použití rozhodlo několik faktorů:

- Snadná implementace - SOAP je dnes již součástí PHP a to jak klient tak i server
- Snadné zpracování dat - data jsou přenášena ve formátu XML, což ve většině programovacích jazyků je jednoduché zpracovat. Navíc v případě PHP se v případě poslání objektu data v cíli sestaví opět do objektu se stejnou strukturou
- Široká rozšířenost - díky své jednoduchosti a využití HTTP protokolu je dnes SOAP velice rozšířený.

Ovšem SOAP má také své nedostatky:

- Díky XML posílá velké množství zbytečných dat
- Parsování XML nepatří mezi nejrychlejší operace
- Každý požadavek je posílán jako samostatný HTTP request - pomalá komunikace

6.1 WSDL

WSDL je jazyk, pomocí kterého se popisují funkce a nestandardní datové typy webových služeb. Je zapsán ve formátu WSDL. Tento soubor slouží klientům připojujícím se k webové službě k tomu, aby věděli, jaké funkce služba poskytuje a v jaké struktuře přijímá a vrací data.

6.2 Tvorba

Toto API má vlastně sloužit jako mezivrstva mezi doteď vytvořeným doménovým modelem a frontendem. Stará se o zabezpečení toho, k jakým datům má daný frontend přístup. Také nahrazuje určitou obecnou logiku, která by se nad získanými daty opakovaně prováděla na jednotlivých eshopech.

Pro rozpoznání toho, jaký frontend daný požadavek na API vykonal, lze využít dvou jednoduchých způsobů. První možností je každé funkci z API přidat dva parametry - APIName a APIPassword, což by ale znamenalo do funkce posílat parametry, které se samotnou logikou funkce nemají nic společného. Druhou možností a zároveň tou, pro kterou jsme se rozhodli, je posílání těchto údajů v HTTP hlavičce. Hlavička se nastaví jednou při vytvoření objektu, který následně volá jednotlivé funkce a který ke každému volání přidá tuto část hlavičky. Díky tomu se nám do parametrů jednotlivých funkcí nepletou žádné logicky nesouvisející parametry.

Pro každá data, která mají být přenášena pomocí API, jsem se rozhodl vytvořit speciální objekt. Je to z důvodu úspory přenášených dat. Mohly by se sice posílat celé instance objektů získané z našeho doménového modelu, ale zdaleka ne všechny informace jsou na

frontendu potřeba - například nákupní ceny klíčů. Také lze zjednodušit strukturu dat - jazykové mutace jsou v doménovém objektu řešeny jako asociativní pole, ale na frontendu v jednu chvíli chci vždy jen jednu jazykovou mutaci - pole není potřeba.

Každá funkce a struktura každého objektu, který bude posílán pomocí API jedním či druhým směrem, musí být popsána v wsdl.xml. Tvořit tento soubor ručně se ukázalo jako prakticky nemožné, protože hlavně při prvotním vývoji API docházelo často ke změnám. Neustálá ruční editace tohoto souboru velice zdržovala, a tak jsem se rozhodl pro jeho automatické generování na základě správně okomentovaných funkcí a vlastností jednotlivých objektů. Pro generování jsem využil volně dostupné části Zend frameworku, která se ukázala v tomto směru velice nápomocnou. Narazil jsem jenom na jeden problém a to, že v případě využívání namespace se musí napsat absolutní adresa k dané třídě. V opačném případě generování skončí chybou.

Automatické generování wsdl.xml pro každý požadavek bylo velice pomalé. Proto generování neprobíhá automaticky, ale jeho volání se provádí ručně po provedení úprav v samotném API. Klienti si potom žádají přímo wsdl.xml, který je již uložen fyzicky na disku.

6.3 Přenos klíčů

Jednotlivé aktivační klíče jsou buď uloženy jako naskenované/vyfocené obrázky na disku nebo jsou generovány z textového zadání. Původní nápad byl posílat adresu zabezpečenou pomocí nějakého hashe, který by se na straně serveru dekódoval a podstrčil obrázek. To se mi ovšem nelíbilo z několika důvodů. Zkoušení náhodných hashů je velice jednoduchý proces a zákazník by se takhle mohl dostat ke klíčům, které mu nepatří. Další věc je, že takové zkoušení by nám zbytečně zatěžovalo servery. Jako řešení jsem zvolil posílat obrázek ve formě data URI scheme, což je způsob, jak přenášet surová data a ne jenom cestu k nim. V našem případě se tedy přenáší informace o tom, jak má obrázek s daným klíčem vypadat, místo adresy, na které je obrázek s klíčem dostupný. Toto přineslo neprůstřelné zabezpečení při zobrazování klíčů konečným zákazníkům. Ovšem má to i své stinné stránky a to je podpora v prohlížečích. K dnešnímu dni nás již naštěstí toto omezení nemusí trápit, protože tuto technologii podporují všechny moderní prohlížeče a Internet Explorer do verze 8 (včetně).

7 Frontend

7.1 Tvorba

Jelikož firma v době mého nástupu již vlastnila svůj eshop Key4You.cz, tak se jednalo ”jen”o vytvoření nového frontendu, který bude vypadat stejně jako dosavadní eshop, ale bude již získávat data pomocí vytvořeného SOAP API.

Samotný eshop měl standardní funkce jako filtrování a řazení produktů, přihlašování a registrace uživatelů, tvorba objednávek, napojení na platební brány, napojení na Google, Heureka a Zboží pomocí xml feedů.

Díky využití Nette se prakticky samy tvořily tzv. pěkné cool URL adresy, kde je adresa tvořena čitelně pomocí / a ne za pomocí ”šíleného ocásku”?parma1=value1&param2=value2... Bohužel díky tomu potom neseděly adresy na feedy a notifikační adresy pro jednotlivé platební brány. Naštěstí Nette pamatuje i na takovéto případy a lze nastavit routy tak, aby se na konkrétní adrese (původní notifikační adresa) zobrazoval libovolný obsah (novou notifikační adresu).

Jiný problém při tvorbě frontendu nenastal. Na rozdíl od původní verze jsem optimalizoval zobrazení webu na všechny v současné době běžně používané prohlížeče a IE až do verze 8.

Po úspěšném vytvoření celého frontendu se přišlo na jednu velice znepokojivou věc. Průměrná doba načtení stránky byla kolem 5 sekund. V některých případech až dvojnásobek.

8 Optimalizace rychlosti

Po prvotním testování se ukázalo, že načítání jednotlivých stránek na frontendu je velice pomalé. Díky komunikaci s API načtení jedné stránky trvalo kolem 5 sekund. Proto jsem musel provést několik optimalizací.

8.1 Omezení komunikace s API

První optimalizace byla provedena na snížení požadavků volání API. V některých případech se na pozadí získával produkt, který byl už jednou načten dříve (ať už v hromadném načítání více produktů nebo v dříve volané funkci). Každou entitu, která má své id (produkt, platformu, vydavatele...), hned po získání z API jsem zapsal do asociativního pole pod hodnotu jejího id. V případě, že chceme nějakou položku získat znovu, tak se nejdříve zkontroluje toto pole a API se využije pouze v případě, že nebyla v tomto poli položka nalezena. Bohužel toto se nedělo moc často a na většině stránek eshopu se tato úprava vůbec neprojevila.

Optimalizace na 5 sekund

8.2 Cachování výsledných dat/šablon

Po dalším zvážení jsem začal cachovat data získaná přes API a jako úložiště jsem použil soubory. Pro toto cachování jsem využil objektu Cache, který je součástí Nette a řeší problémy jako invalidace po časové expiraci nebo zajištění toho, že se nepokusí dvě vlákna současně zapisovat do jednoho souboru.

Pro dosažení co největší časové úspory jsem chtěl cachovat na co nejvzdálenější vrstvě od samotného API. Ideálním místem pro většinu případů je cachovat až výsledné HTML, které získám ze šablonovacího systému latte (standardní šablonovací systém Nette). Ovšem ne vše lze cachovat až v šabloně. Například počty produktů, potřebné pro sestavení stránkovače, se do šablony nikdy nedostanou. V těchto případech cachuji data na úrovni presenteru.

Každá data v cachi mají nějaký svůj klíč, pod kterým jsou přístupná. Protože se na webu zobrazují různé informace (hlavně ceny) pro obyčejné zákazníky, premium zákazníky a velkoobchodníky, tak do klíče se musí kromě URL adresy, pro kterou je daná cache sestavená, přidat také parametr, který rozhodne, pro který typ uživatele je dané cache sestavená.

Necachuji celé výsledné HTML, ale jen určité jeho části. Díky tomu mohu mít například banner v cachi pouze jednou pro daný typ zákazníka a potom této cache využít na všech stránkách, kde se má daný banner zobrazit.

Jako časové intervaly pro expiraci dat jsem zvolil 1 minutu a 1 hodinu. Všechna místa, kde je zobrazena cena, jsou v cachi pouze 1 minutu. To je z důvodů často se měnících akcí, které mnohdy trvají jen krátkou dobu. Ostatní věci (abecední seznam produktů, seznam kategorií...) jsou udržovány v cachi po dobu jedné hodiny.

Optimalizace na 300 milisekund

8.3 Tvorba odkazů v hromadných seznamech

Odkazy v Nette neodkazují na konkrétní adresu, ale na presenter a akci, případně signál a toto se pomocí nastavených rout přeloží na výslednou URL adresu. Každý odkaz vytvořený v Nette musí projít všemi routami, dokud nenarazí na takovou, která ho dokáže přeložit. Díky výše popsanému způsobu zachování starých notifikačních a feed adres, výrazně vzrostlo množství rout v aplikaci - z původních 3 na 15. Kdyby se mělo přeložit jen několik desítek odkazů, tak je toto zpomalení prakticky neměřitelné. Bohužel na frontendu jsou dva abecední seznamy, které obsahují kompletně všechny produkty. Dohromady tyto dva seznamy mají cca 2000 odkazů.

Jako řešení tohoto problému jsem využil toho, že všechny odkazy jsou stejné až na id a název produktu v adrese. Proto jsem si nechal vygenerovat jenom jeden přeložený odkaz od routovacího systému Nette s placeholdrama na místě id a názvu produktu. Potom jsem vytvářel odkazy na tuto adresu, kde jsem vždy placeholdry nahradil hodnotami pro konkrétní produkt.

```
{var $link = $presenter->link('Product:details', array('id'=>'__id__', 'name'=>'__name__'))}
<ul>
    {foreach $al as $key=>$products}
    <li>
        <a href="#">{$key}</a>
        <ul class="bg-silver R1 R2 R3 R4">
            {foreach $products as $product}
            <li>
                <a href="{str_replace(array('__id__','__name__'),array($product->id, $template
                    ->webalize($product->title)), $link)}">{$product->title}</a>
            </li>
            {/foreach}
        </ul>
    </li>
    {/foreach}
</ul>
```

Výpis 3: Placeholdery při tvorbě odkazů

Díky tomu je vygenerování těchto seznamů cca o 300 milisekund rychlejší. Protože ale tyto seznamy jsou následně cachovány po dobu jedné hodiny, tak zrychlení je naprosto zanedbatelné.

Optimalizace na 300 milisekund

8.4 Vytváření SOAPClient objektu

Každý typ dat získávaných z API má svůj vlastní Repository soubor, stejně jako v případě doménového modelu na úrovni administrace. Každý tento Repository má svůj vlastní SOAPClient objekt, který se vytváří v jeho constructoru. Bohužel ihned po vytvoření se připojuje na server se službou a žádá si wsdl.xml. Toto jsou zbytečné požadavky, které získávají stejná data. Proto jsem vytvořil SOAPClient jako singleton a ještě ho vytvářím pomocí lazy loadingu až v momentě, kdy chci volat první požadavek na API.

Optimalizace na 60 milisekund

9 Uplatněné a získané dovednosti a znalosti

Protože se programování i tvorbě webů věnuji již od deseti let a navíc jsem si mohl vybrat libovolné technologie, které mně osobně sedí a které znám, tak jsem většinu potřebných znalostí již měl, například o PHP, MySQL, OOP, Nette... Ovšem úplně nové pro mě bylo vše kolem technologie SOAP. Nějaké podvědomí jsem o ní sice měl, ale spíše jenom to, že něco takového existuje. Praktické zkušenosti jsem neměl žádné. Funkční principy této technologie a její používání jsem musel sám nastudovat. Informace jsem získal na internetu a v odborné literatuře. Takže přínosem pro mne bylo seznámení se s touto technologií a vyzkoušení jejího použití v praxi. To byl největší přínos z této odborné praxe.

Na druhou stranu ještě nikdy jsem nepracoval na projektu takového rozměru a už vůbec ne od začátku. Díky tomu jsem si mohl vyzkoušet, jak to vypadá v praxi při tvorbě něčeho většího. Toto je rozhodně velice přínosná zkušenost.

10 Závěr

V této práci jsem se snažil popsat své působení ve firmě Aktiv Communication s.r.o. po období vykonávání mé bakalářské praxe. Jsem rád, že jsem mohl svou praxi vykonávat v této firmě a následně u ní dále pracovat.

Systém, který jsem navrhl a vytvořil, nyní úspěšně běží a je stále vyvíjen. K dnešnímu dni je na něj napojeno 8 různých eshopů.

Na žádost majitele firmy jsem tento systém rozšířil o mnoho dalších funkcí, jako například možnost vytvářet dárky k objednávkám, zcela nový typ víkendových akcí, objednávky k potvrzení, generování faktur pro klienty. Největším rozšířením je rozesílání newsletterů, kde lze filtrovat seznam zákazníků, kteří mají daný newsletter obdržet podle mnoha kritérií, například dle počtu proklikutý v daném intervalu, podle typu zákazníka, podle hodnoty, kterou zákazník v eshopě utratil, či podle nakoupeného zboží.

11 Reference

[1] Grúdl, David *Dokumentace — Nette Framework,*
Dostupné 12. 3. 2014 na: http://doc.nette.org/cs/2.1/

[2] The PHP Group *PHP: Hypertext Preprocessor*,
Dostupné 12. 3. 2014 na: http://www.php.net/

[3] LUKE WELLING, Laura Thomson. *PHP and MySQL web development*. 5th ed. Boston, Mass: Addison-Wesley, 2013. ISBN 978-032-1833-891.

[4] FORD, Andrew. *Apache 2: pocket reference*. Sebastopol: O'Reilly, 2008, XII, 195 s. ISBN 978-0-596-51888-2.

[5] LIU, Cricket a Paul ALBITZ. *DNS and BIND*. 5th ed. Beijing: O'Reilly, 2006, 616 s. ISBN 05-961-0057-4.

[6] MCFARLAND, David Sawyer a David Sawyer MCFARLAND. *JavaScript*. 2nd ed. Sebastopol, Calif.: O'Reilly, 2011c2012, xvi, 518 p. Missing manual. ISBN 14-493-9902-9.

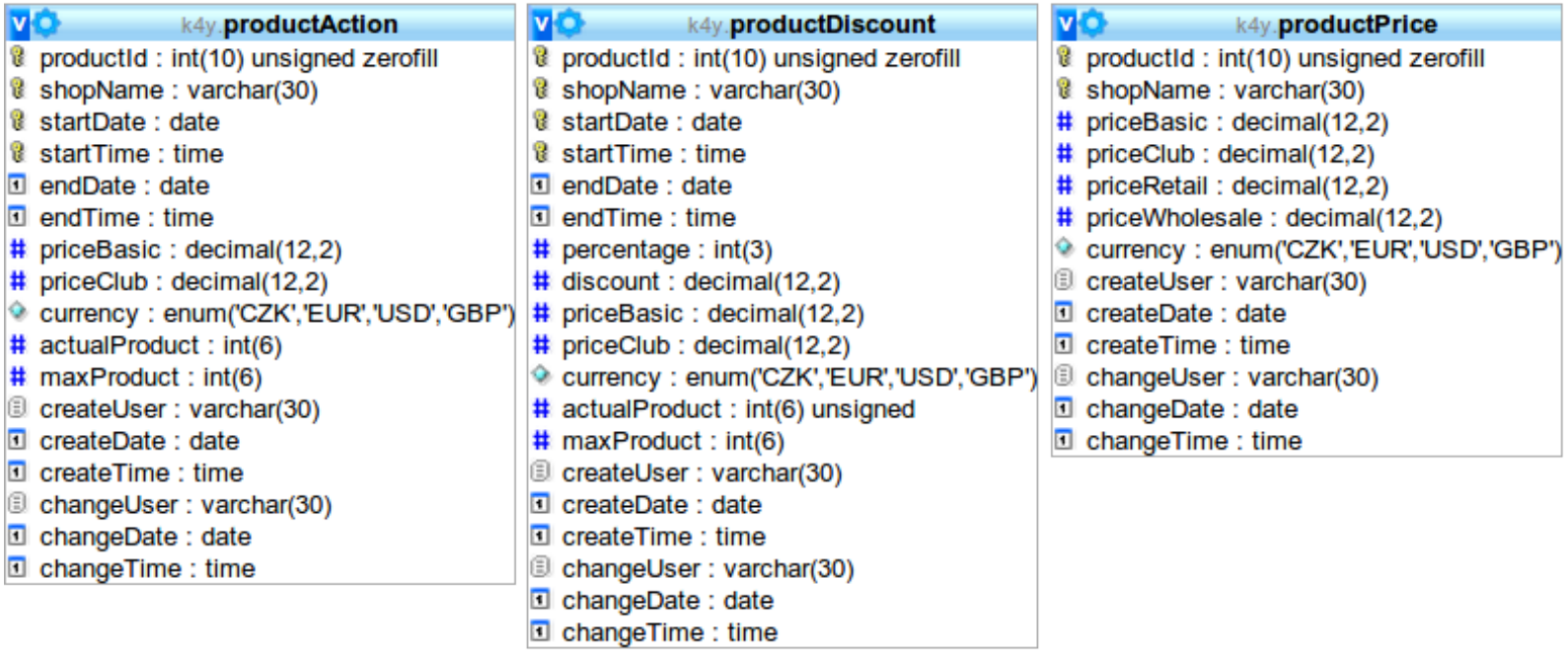
[7] GOLDBERG, Kevin Howard a Elizabeth CASTRO. *XML*. 2nd ed. Berkeley, CA: Peachpit Press, c2009, xviii, 269 p. Visual quickstart guide. ISBN 03-215-5967-3.

[8] DAIGNEAU, Robert. *Service design patterns: fundamental design solutions for SOAP/WSDL and RESTful Web services*. Upper Saddle River, NJ: Addison-Wesley, c2012, xxv, 321 p. Addison-Wesley signature series. ISBN 03-215-4420-X.

A Ukázka struktury databáze

Níže je uvedena ukázka asi nejzajímavější části databáze. Jedná se o struktury tabulek, které uchovávají informace o produktech.

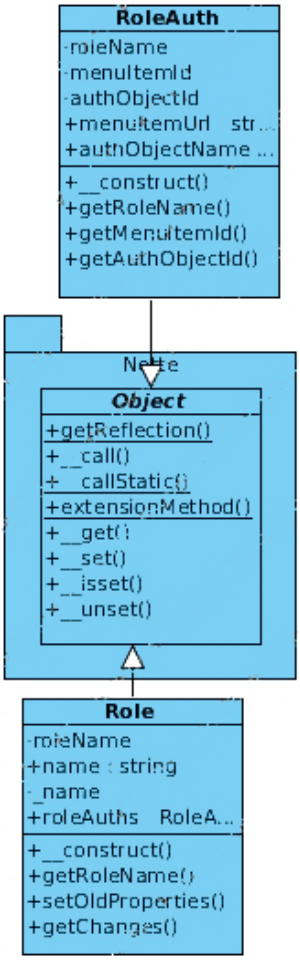
Obrázek 4: Třídní diagram 1



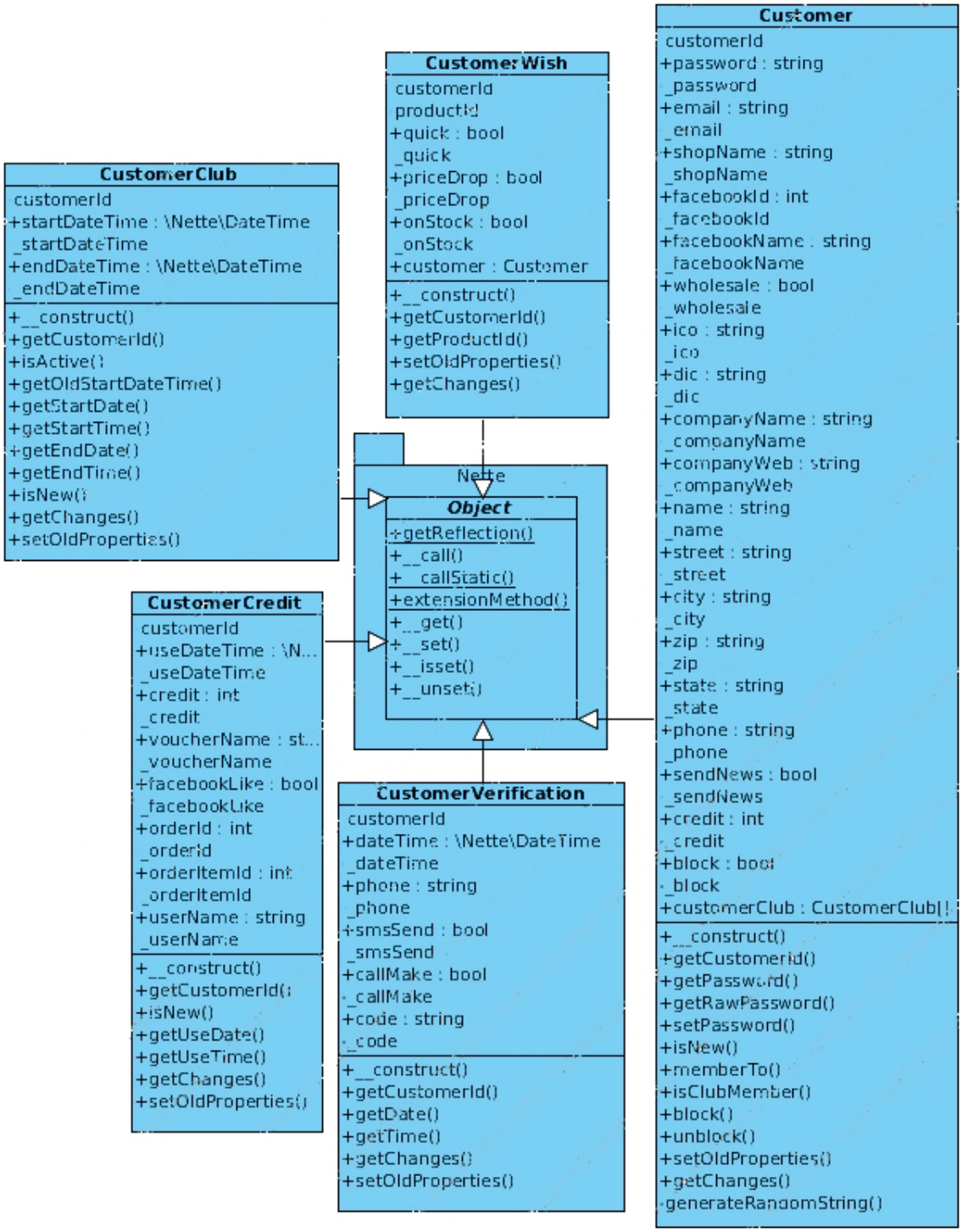
Obrázek 5: Struktura databáze - produkt - ceny

B Entity doménového modelu

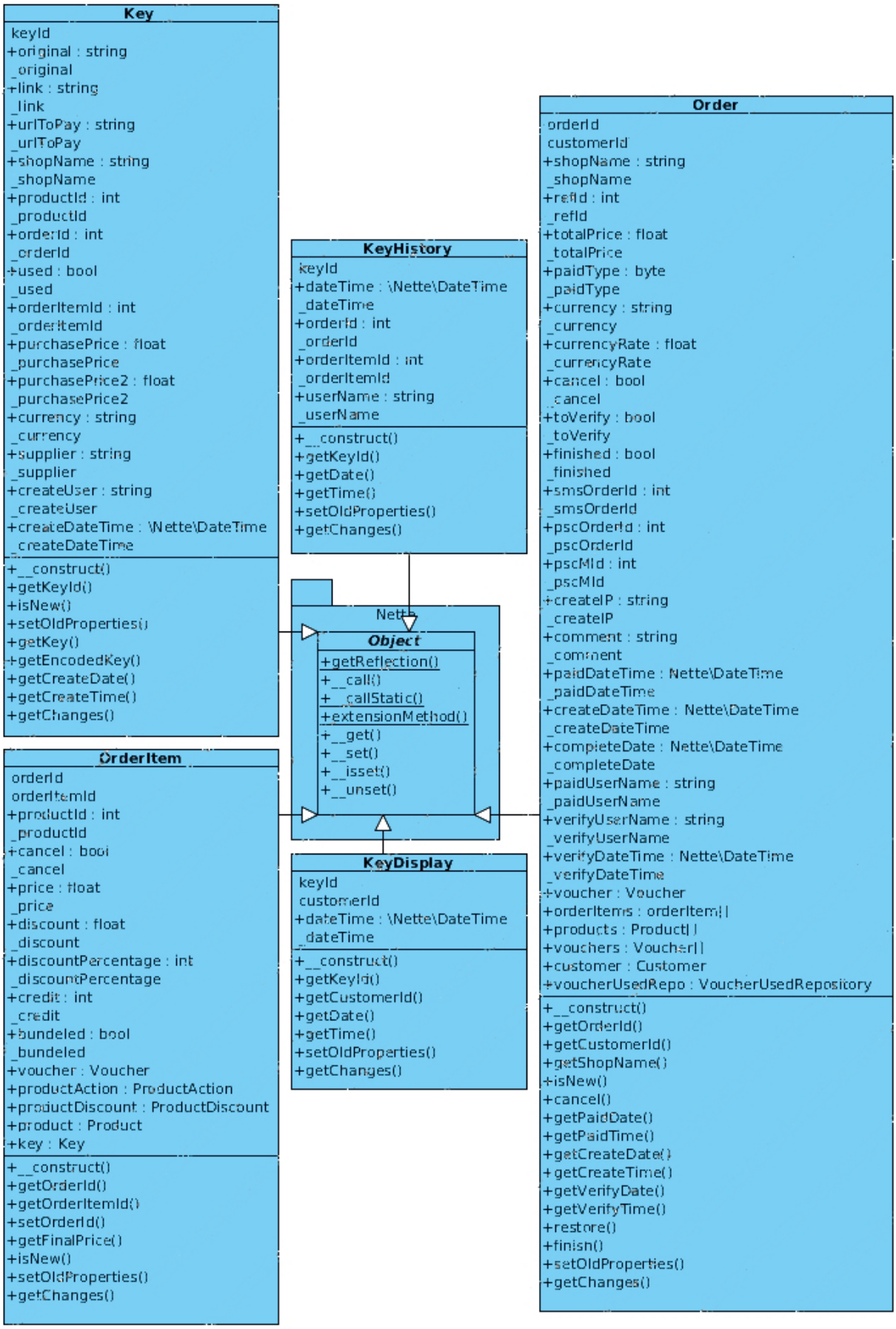
Níže jsou k vidění třídní diagramy všech entit z doménového modelu.



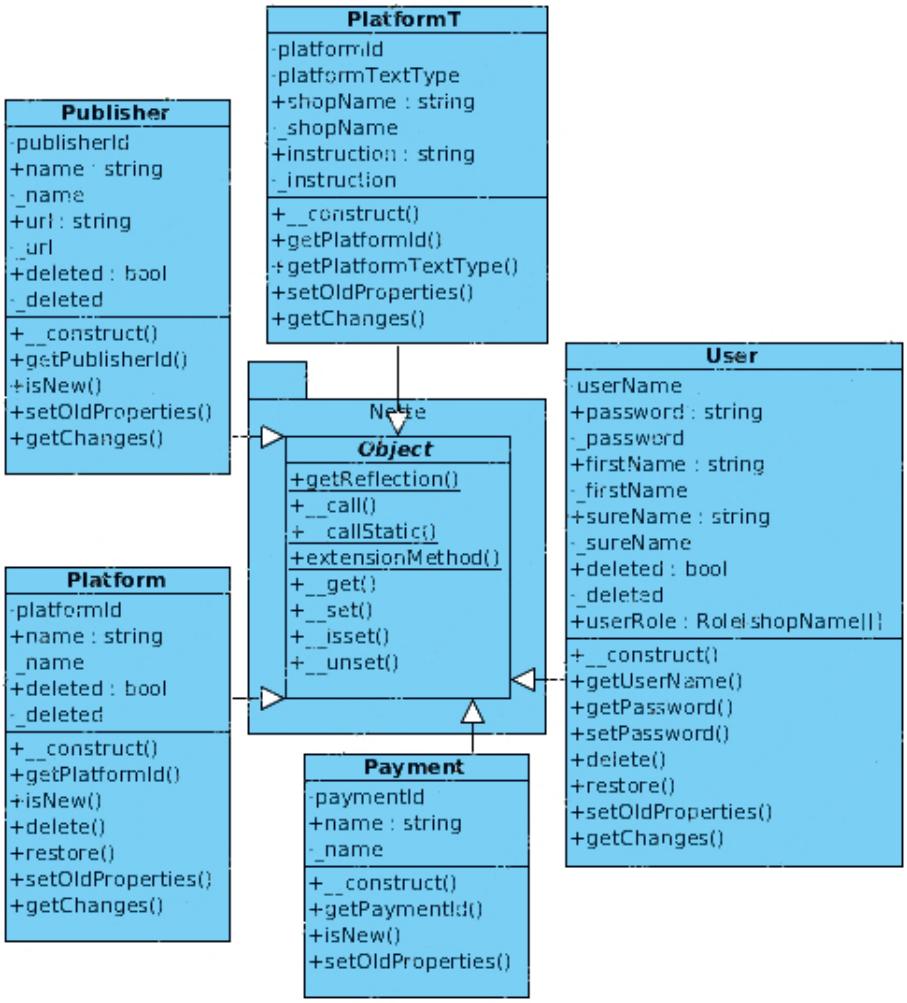
Obrázek 6: Třídní diagram 1



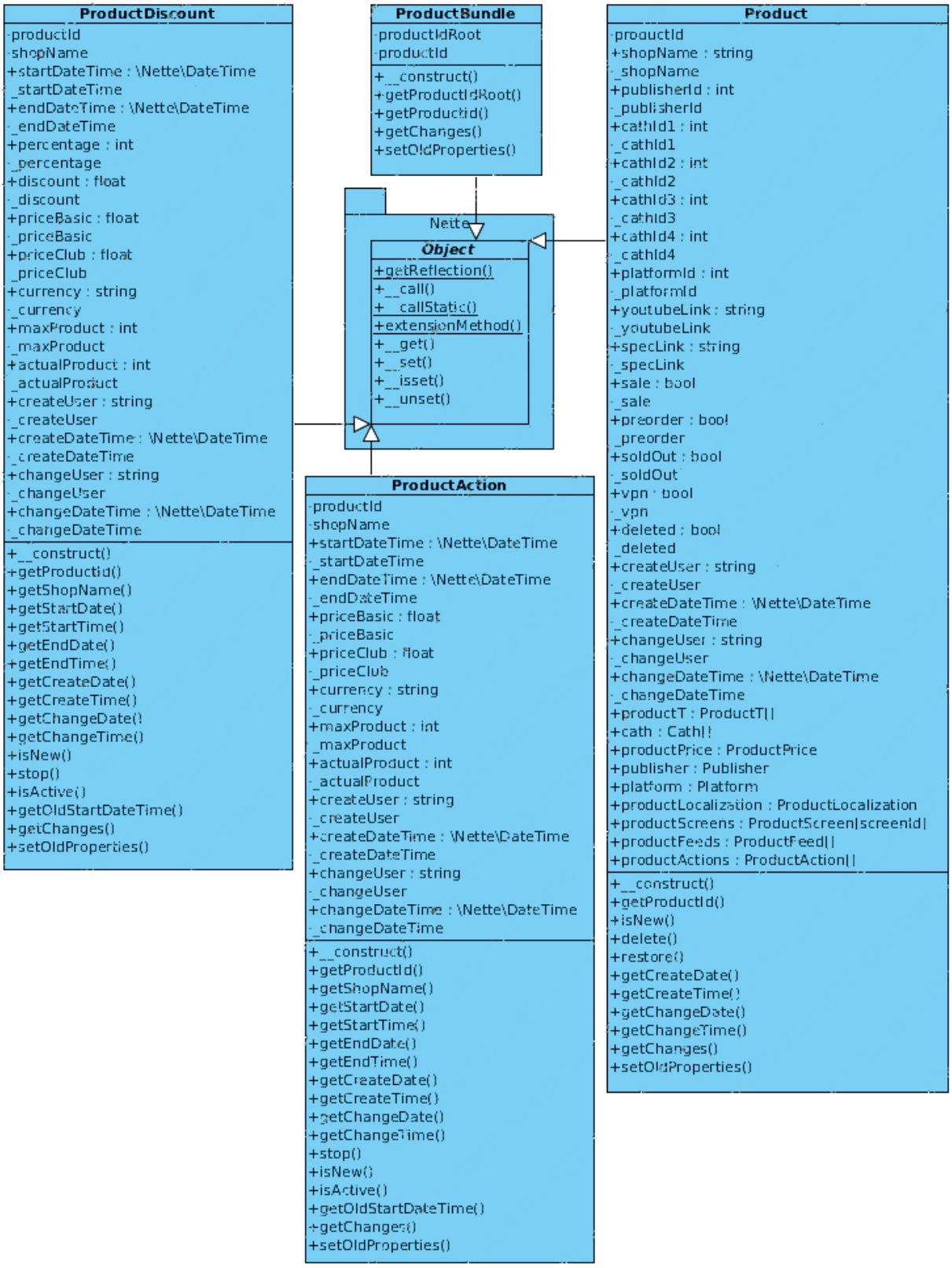
Obrázek 7: Třídní diagram 2



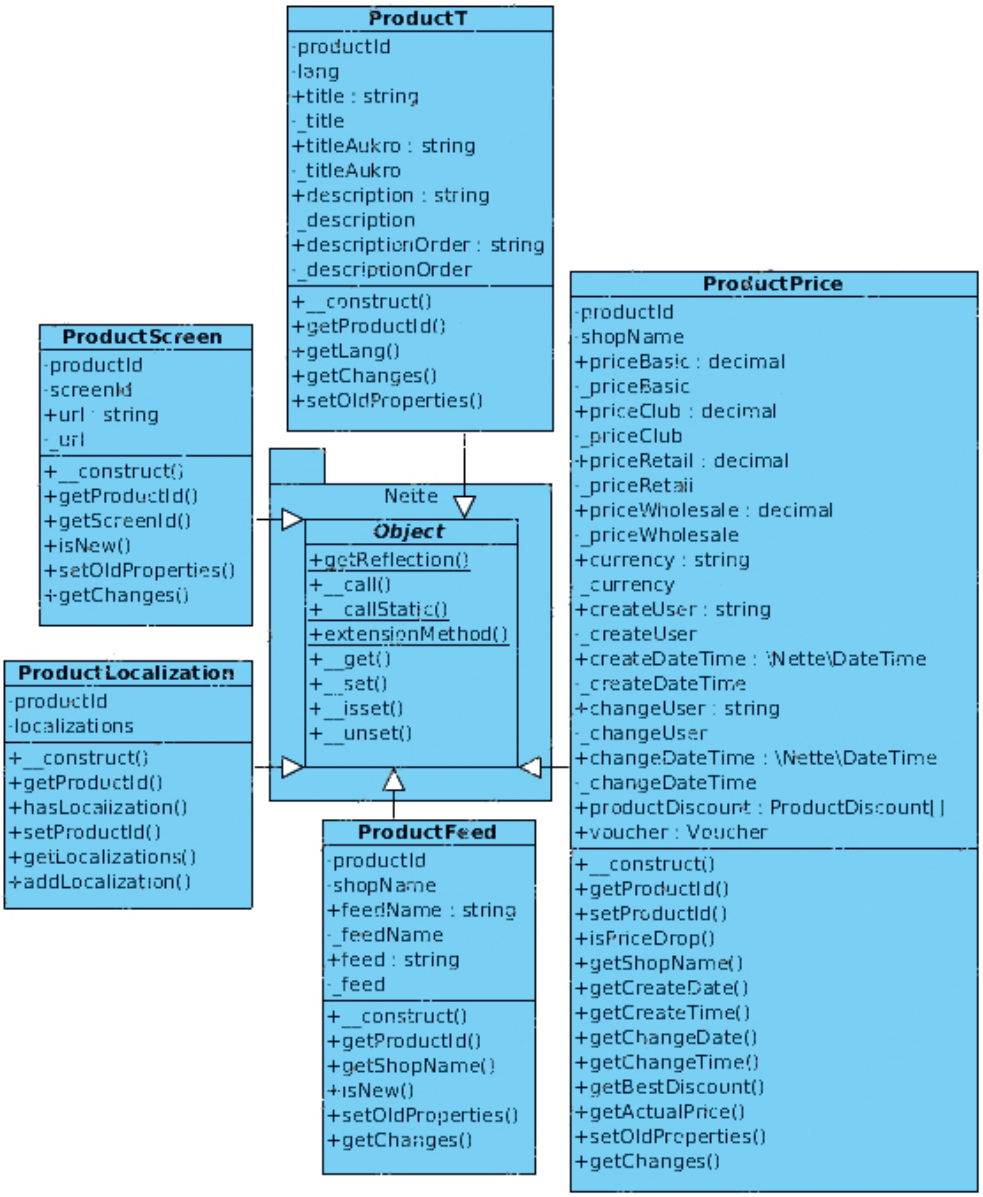
Obrázek 8: Třídní diagram 3



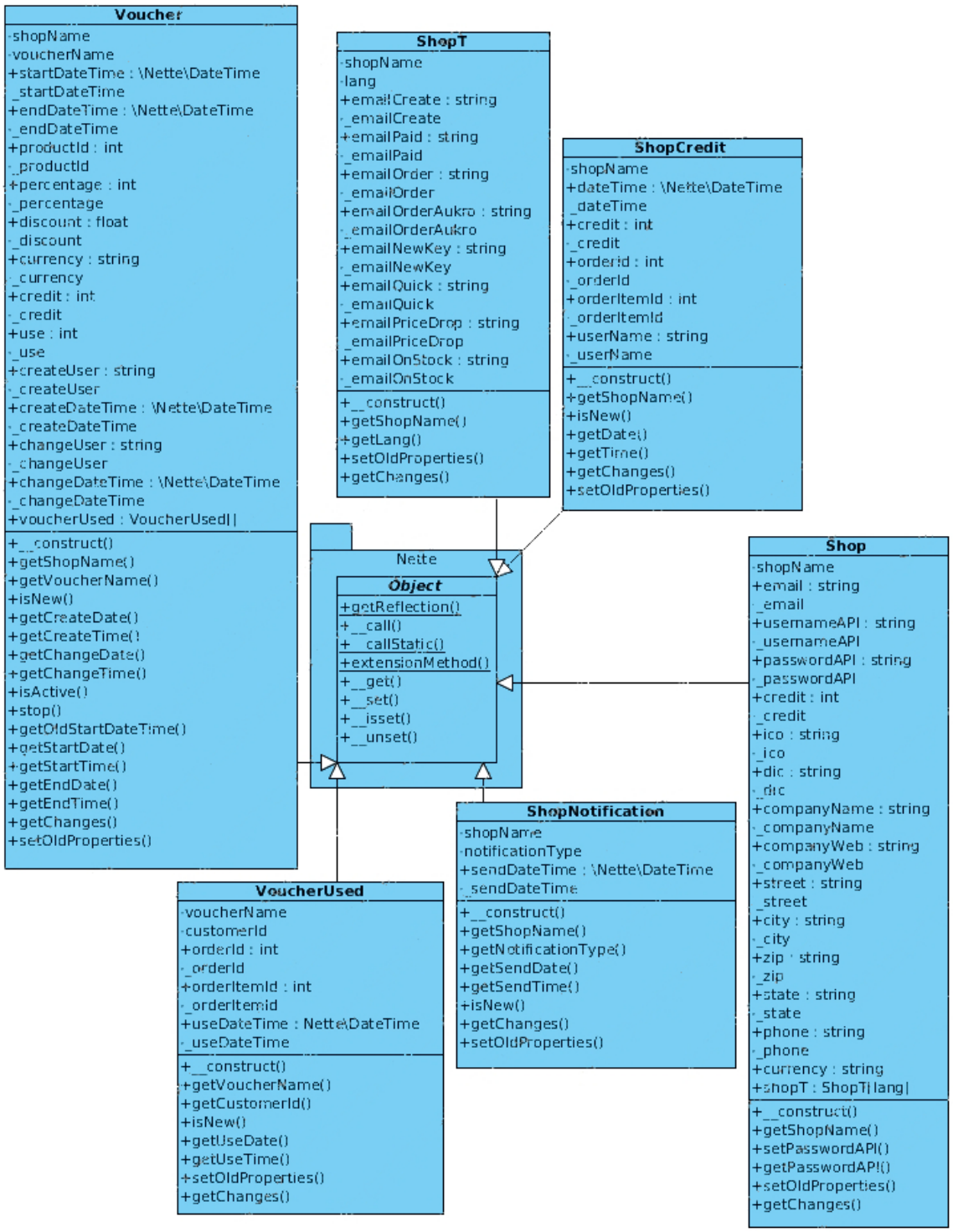
Obrázek 9: Třídní diagram 4



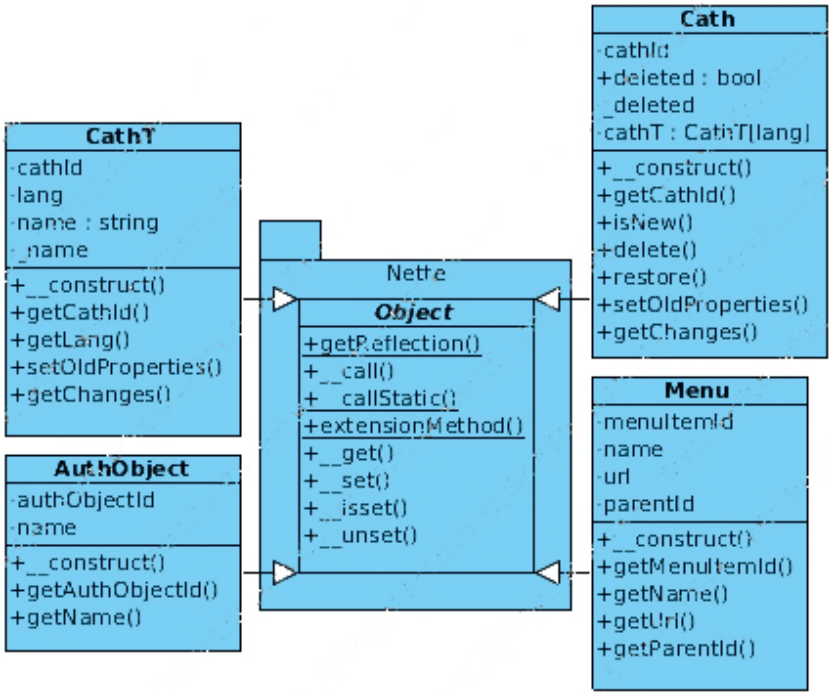
Obrázek 10: Třídní diagram 5



Obrázek 11: Třídní diagram 6



Obrázek 12: Třídní diagram 7



Obrázek 13: Třídní diagram 8